SASE LABS Workshops

MATLAB: A Guide to the Basics & Language Fundamentals

Written By Abi Chiaokhiao

Files: W1_StartFile.m

Disclaimer: Many examples taken from mathworks.com

Table of Contents

The Basics

Grammar

- If you use a semicolon at the end of a line, that line's calculation won't output
- To format outputs, use the `format` keyword (more here)
  - Example: `format long` ⇒ outputs have a long decimal
- To represent imaginary parts of numbers, write `i` or `j` on the end of the number: `4i`
- One line comments use `%`, and what follows is a comment (for multiple: use `%{` and `%}`)

Example

```
x = 2;  %This is a one-line comment
%{
This is a multiple
line comment
%}
```

Variables

In order to make our code more legible and efficient, we use variables. This sets a keyword to a certain value so that it can be used and changed throughout the code.

- How to: `keyword = value`
  - Then press `enter`

Whenever you use *`keyword`*, it's like typing *`value`* but it's easier to understand the role of that value.

- If there are multiple output arguments, enclose them in square brackets
  - [outArg1, outArg2]

Workspace Variables

- The workspace has variables that you create, to see what is currently in the workspace, use the command `whos`

Arithmetic & Common Functions

- For basic arithmetic, you can use their respective operators: `+ - / * ^`, more here
- To use trig functions, write what would be typed in a calculator (more at the end)
  - `cos(val)` & `sin(val)` are two functions
- Another common one is `sqrt(val)` to find the square root of a value
- To find mins or maxs in one or multiple sets of data, there is `min` & `max` (more in Functions)

Other

- For documentation for a function: `doc function` (or `help function` for shorter)
- To use libraries from other languages, go here
- To clear command window: type `clc`

Data types

- Note: MATLAB is a language that does automatic type assigning, so it's different than languages like c++ where the type needs to be denoted

Numbers (more here)

- Include signed and unsigned integers, and single-precision and double-precision floating-point numbers. All are stored as double-precision floating-point

Strings (more here)

- Enclose the text in double-quotes: "*Hello, Earth*", this is a string
    - Want quotes in your text? Use 2 double quotes
        - "*My friend said ""Hi"" to you*"
- To add to the end, use the addition operator (+)
- To find the length of a string do: `strlength(`*`stringName`*`)`
    - If you have a matrix of strings, you will get an array of lengths back

Characters/Character Arrays (more here)

- Text can be represented by an array of characters, in case you want to be able to separate each character, like in dna: `seq = '`*`GCTAGAATCC`*`';`
- To access a certain character, do: *`name`*`(`*`place#`*`)`

Concatenation

- Done inside square brackets: `[`*`charArr1 charArr2`*`]`

Example

```
a = 'abc';
b = 'cde';
[a b]
```

Output: `'abccde'`

Special characters

Some characters you can't just type out to represent them

Example

```
exChar = '''';
disp(exChar)
```

Output: `'`

- Note: `disp(`*`val`*`)` displays the value of *`val`* in the command window
    - More special characters at the bottom of this page.

Conversion

- There is a list of functions here, wherein you put in the value you want to convert and it outputs the converted value

Matrices (Note: the usual notation, and what I will be using, is rows x columns)

Declaring

A matrix is written between one pair of `[]`, where each row is separated by a "`;`" and each element is separated by a space.

Example: 3 x 4 declaration

```
matrix34 = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

- To get a random `n` x `n` matrix, call `rand(n)`
- There's also a "magic" `n` x `n` matrix called with `magic(n)`

Making a matrix of zeros or ones

Use function `zeros` ⇒ `zeros(`*`rows, columns`*`)`

Use function `ones` ⇒ `ones(`*`rows, columns`*`)`

The Inverse of a Matrix

Use function `inv(`*`matrix`*`)`

Arithmetic

If you take the matrix name and use arithmetic, all elements in that array will have that math applied to them. Same for trig functions (just do trigFunction(matrixName)).

Example

```
a = [10 15; 25 40]
a/5
```

Output:

```
a = 2 3
    5 8
```

Matrices can be used together (matrix1*matrix2 ⇒ matrix multiplication). If you want element-wise arithmetic, you need to use a period before the operator.

Example

```
a = [3 4; 1 2];
b = [2 5; 1 8];
c = a*b
d = a.*b
```

Output:

```
c = 10 47
     4 21
d =  6 20
     1 16
```

Matrices (con.)

Concatenation: Joining arrays to make bigger ones

- This first in MATLAB this is done by adding respective rows of the second array to the ones of the first

Example

```
a = [1 2; 3 4]
A = [a,a]
```

Output:

```
A = 1 2 1 2
    3 4 3 4
```

- To add the matrix as additional rows use a semicolon instead of a comma

Example

```
a = [1 2; 3 4]
A = [a;a]
```

Output:

```
A = 1 2
    3 4
    1 2
    3 4
```

Use

Accessing elements

`matrixName(row,column)`

or

`matrixName(place)`

- Can use to check values or change them
- The place is determined as if you were reading English (left to right, top to bottom)
    - If you assign a place that doesn't exist, the matrix will increase to accommodate

Example

```
a = [3 4; 1 2];
a(3,3) = 20
```

Output:

```
a =  3 4  0
     1 2  0
     0 0 20
```

- To access multiple elements, use a colon
    - `matrixName(1:3,2)` ⇒ 2nd elements of 1st through 3rd rows
    - Just using the colon would indicate all elements of that dimension

Other Notes

- For those that have taken Multivariable, to transpose the matrix, it is **`matrixName'`**
- Another way to use the colon is for initializing vector values equally spaced apart
    - `0:5:20` ⇒ `0 5 10 15 20`
- More documentation for declaring, formatting, and indexing are [here]

2-D Line Plots

Initializing

- Use `plot(xVals,yVals)`
  - Unless you just want a single point, xVals & yVals need to be functions that determine multiple values
  - For x values, a nice function is `linspace(a,b,n)`, which makes `n` points from `a` to `b`, if no `n` is given, 100 points are made

Labeling (written after initializing the plot)

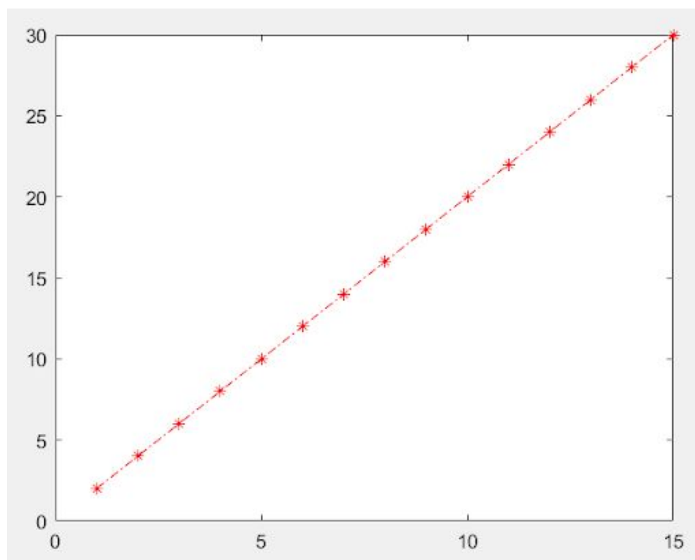
- x-axis: `xlabel('x')`
- y-axis: `ylabel('sin(x)')`
- The whole plot: `title('Plot of the Sine Function')`

Formatting

To print out properties, assign the plot to `ln`: `ln = plot(x,y)`

To format, add a 3rd argument when initializing the plot. The order of the characters doesn't matter unless they're for the same specification (then they have to be together).

Markers have attributes you can change: `MarkerSize`, `MarkerEdgeColor`, `MarkerFaceColor`, `MarkerIndices` (placement)



Example

```
x = 1:15;
y = 2*x;
plot(x,y,'r-.*')
```

Output to the right

<u>r</u> ⇒ red color

<u>-.</u> ⇒ a dash-dot line

<u>*</u> ⇒ points are labeled with stars

More formatting options [here]

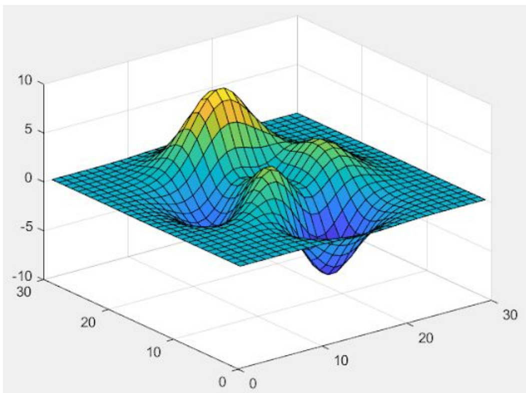
Using Multiple Plots

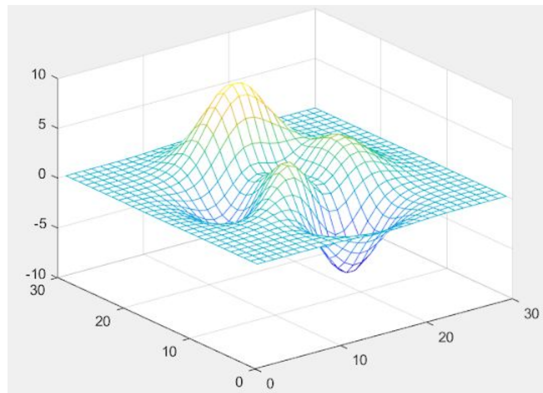
- Use command `hold on`, the following plots will be added to the original
- To end adding onto the plot use `hold off`

3-D Surface Plots

Initializing

1. Create domains for x & y: `[X,Y] = meshgrid(start:step:max);`
2. Create a function for z (preferably dependent on x & y)
- Another way to make a surface as a test is to use the function `peaks(n)`
3. Use `surf` or `mesh` to plot: `surf(x,y,z)`, `mesh(x,y,z)`
    a. `surf` (left) colors the surface & the connecting lines, `mesh` (right) colors the lines, the rest of the surface is an opaque white





Displaying Multiple Surface Plots

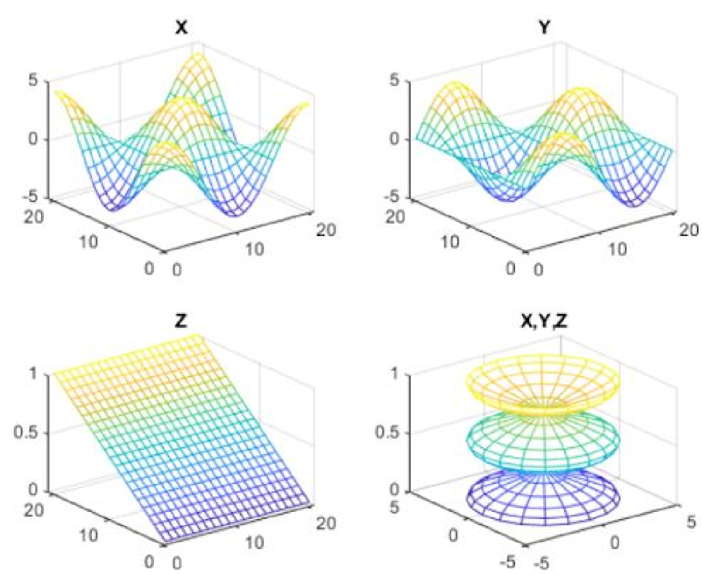
You can display multiple on the same page (not on the same graph like 2-D plots

- The function is `subplot(m,n,p)`, `m` & `n` determine the arrangement (`m` x `n` matrix), and `p` is the number of placement (left to right, top to bottom)
- There's an optional 4th argument for formatting and editing, link in the Functions table

Example

```
t = 0:pi/10:2*pi;
[X,Y,Z]=cylinder(4*cos(t));
subplot(2,2,1);
mesh(X); title('X');
subplot(2,2,2);
mesh(Y); title('Y');
subplot(2,2,3);
mesh(Z); title('Z');
subplot(2,2,4);
mesh(X,Y,Z);
title('X,Y,Z');
```

Output:



Loops

Loops repeat sections of code you allocate for as many times as their conditions are met. The two we will cover are `for` and `while` loops.

- Note: `end` keywords must align with their respective starting keywords

`for` Loops

The keyword `for` starts this loop, it is followed by a condition with a predetermined number of iterations. The following code is allocated by starting after the line that includes `for` and the condition, and going until it hits the `end` keyword.

Example

```
for cats = 1:5
    bowls(cats) = cats*2;
end
bowls(1:5)
```

Output: `2  4  6  8  10`

`while` loops

The keyword `while` starts this loop; it is followed by a condition (number of iterations is not known). The following code is allocated by starting after the line that includes `while` and the condition, and going until it hits the `end` keyword.

Example

```
peeps = 3;
greeting = "Ayy wassup";
hellos = 0;
while hellos < peeps
    disp(greeting)
    hellos = hellos + 1;
end
```

Output:

```
Ayy wassup
Ayy wassup
Ayy wassup
```

Other documentation for loops [here]

Conditional Statements

Conditional statements run (once) a specific section of code you allocate, but only if the written condition is met. The two we will cover are `if` and `switch` statements.

- Note: `end` keywords must align with all respective starting keywords

`if` Statements

Starts with `if` which is followed by a condition(s). If met, the allocated code will run once. If not, the computer will move on. The code is allocated by starting after the line that includes `if`, `elseif`, or `else`, and goes until it hits another one of those or the `end` keyword.

`elseif (`*`condition`*`)`: Followed by code that's run if the condition is met. Needs to be preceded by an `if` statement.

`else`: Followed by code that's run if no previous conditions are met. Needs to be preceded by at least an `if` statement. May also follow one or more `elseif` statements.

Example

```
i = 5;
if i < 0
     str = "That's negative";
elseif i > 0 && i < 10
    str = "That's a nice number";
else
    str = "Eh, idk if I like that number";
end
disp(str);
```

Output: `That's a nice number`

Note: "`&&`" is the and operator. This denotes additional conditions to be followed as well as the preceding one(s). There is also an or operator ("`||`"). More logical operators at the top of this page

Conditional Statements (con.)

`switch` Statements

Starts with `switch` which is followed by an expression. It's evaluated once and runs specific code based on if a case value is equivalent. Denoted with keywords `case` or `default`.

`case value`: There can be multiple of these. Followed by code that's run if the expression is equivalent to `value`.

`otherwise`: There can only be up to one of these. Followed by code that's run if the expression doesn't equal any of the case values. Preceded by all cases.

Example 1: Comparing values

```
siblings = 4;
switch siblings
     case 0
          output = "Alright only child";
     case 1
          output = "Dos children I see";
     case 2
          output = "Average sized family here (if you
     round)";
     otherwise
          output = "Oh a big family here";
     end
     disp(output)
```

Output: `Oh a big family here`

Example 2: Comparing Ranges or Against Conditions

```
siblings = 10;
switch true
     case siblings < 0
          output = "Wait what";
     case siblings > 2 && siblings < 7
          output = "This is higher than average";
     case siblings > 7
          output = "Wth";
     otherwise
          output = "You have an average family";
     end
     disp(output)
```

Output: `Wth`

Functions (pg 1/3)

In Document

Input	Output	Description	Section
`format val`	Outputs	Outputs following this line are changed according to the `val` value. Formats here	Basics
`whos`	Workspace variables	The Name, Size, Bytes, Class, and Attributes for each existing workspace variable are displayed in a table	
`cos(val)`	Calculated value	The cosine of `val` is returned. Other trig functions (cotangent, cosecant, etc) here	
`sin(val)`	Calculated value	The sine of `val` is returned. Other trig functions (cotangent, cosecant, etc) here	
`sqrt(val)`	Calculated number	The square root of a given `val` is returned	
`min(a)` or `min(a, b, ...)`	A number or numbers	In the data `a, b, ...`, the min value (1 argument) or values (multiple arguments) are returned. More here	
`max(a)` or `max(a, b, ...)`	A number or numbers	In the data `a, b, ...`, the max value (1 argument) or values (multiple arguments) are returned. More here	
`doc function`	A window	The documentation for the given `function` is displayed in full in a window	
`help function`	Text	A shortened version of the documentation for the given `function` is displayed in the command window	
`clc`	N/A	Clears the command window	
`strlength(strName)`	A number	The length of a given `strName` is returned	Data Types
`name(place#)`	A single character	The `place#` element of a given character array of `name` is returned	
`rand(n)`	A matrix	A matrix `n` x `n` of random numbers is returned (more uses here)	Matrices

Functions (pg 2/3)

Input	Output	Description	Section
`magic(n)`	A matrix	An n x n matrix is returned where the elements are 1 to n^2 and the sums of the rows and columns are equivalent. n needs to be at least 3 to be valid	Matrices
`zeros(#rows, #columns)`	A matrix	A matrix is made where every element is 0 with that many rows and columns	
`ones(#rows, #columns)`	A matrix	A matrix is made where every element is 1 with that many rows and columns	
`inv(matrix)`	A matrix	The inverse of a given `matrix` is returned	
`plot(xVals, yVals, specs)`	A plot	A 2-D line plot. x values are determined by the function `xVals`, and y values are determined by the function `yVals`. `specs` are optional to format the plot	2-D Line Plots
`linspace(a,b,n)`	A row vector	A vector of n elements is made evenly spaced from a to b. n is optional, and if not included is 100 by default	
`xlabel('x Ax')`	Title	Labels the x-axis of a 2-D plot with the given `x Ax`	
`ylabel('y Ax')`	Title	Labels the y-axis of a 2-D plot with the given `y Ax`	
`title('Plot Title')`	A character array (as a title)	Labels a 2-D plot with the given `Plot Title`	
`ln = plot(x,y)`	Properties	Commonly used properties of the given plot are displayed	
`hold on`	A plot	Following 2-D plots are graphed onto the preceding plot until `hold off` is used	
`hold off`	A plot	No more 2-D plots are added	

Functions (pg 3/3)

Input	Output	Description	Section
`[X,Y] = meshgrid(min:step:max)`	A 2-D grid	A 2-D domain is created where both dimensions have a starting point of *min*, a difference between each point of *step*, and an ending value of *max*	3-D Surface Plots
`surf(x,y,z)`	A 3-D surface plot	A 3-D surface is graphed and returned. The points for all three dimensions are given in *x*, *y*, and *z*. More here	
`peaks(n)`	A 3-D surface plot	A 3-D surface is graphed and returned from a matrix. The matrix has a dimension of *n* x *n*. The values are a test data set that explores a range of values.	
`subplot(m,n,p); graph;`	A plot	A grid *m* x *n* where each element is a plot, *p* is the position *graph* goes into (other calls to the function are needed for the remaining positions). More here	